A Prototype Distributed Visualization System

Part I: Implementation of Prototype System

Contents

1 Introduction **2**

2 Distributed Visualization Environment **2**

2.1 Objectives 3

2.2 Implementation Criteria 4

2.3 Objective Function 4

3 Prototype System **5**

3.1 Operation Modes 5

3.2 Parallel Implementation 6

3.3 Application Program Interface 7

4 Performance Analysis **8**

4.1 Compression Parameters 8

4.2 Parallel Versus Sequential 9

1 Introduction

The computational power of a centralized high performance computing facility cannot be fully appreciated by the remote user without an adequate data transmission capability to transfer the processed data in a timely manner. Network congestion is expected to persist, based on the fact that both the growth of the user community and advances in processor technology have continuously been ahead of advances in network technology. The transmission bottleneck currently precludes the use of distributed visualization applications which involve the interactive display of large volumes of data. The objective of this task is to provide a near-real-time (several frames per second) visualization capability to remote users by employing on-line data compression/decompression technology.

Distributed Visualization involves data flow between a visualization server computer and a client computer over a global network. When the network transfer rate is low or the network congested, the data transfer time can be too long (over a minute for a 512×512 image frame) to support an interactive data processing environment. In order to reduce the data transmission time, data volume must be reduced significantly.

A significant reduction in data volume cannot be achieved, however, without a reduction in data quality. Further, processing time for data compression/decompression becomes additional overhead. Thus the compression algorithm must be computationally simple and provide both a high compression rate and low data-quality degradation. A reversible compression method—Efficient Reversible Image Compression (ERIC)—was developed to satisfy this requirement. ERIC provides lossy as well as lossless modes. In the lossless mode, the compressed data can be transferred for any arbitrary compression rate; in the lossy mode, the compression rate cannot be precisely controlled. A prototype system was developed using ERIC, and a comprehensive performance benchmark was made. A detailed description of the ERIC algorithm is presented in Part II.

This report examines the necessary and sufficient conditions for achieving the objective of distributed visualization, presents a trade-off analysis formula, discusses a parallel prototype system implementation, and validates the performance of the prototype system according to the trade-off analysis formula presented herein.

2 Distributed Visualization Environment

Employing data compression to reduce transmission time introduces computational overhead and data quality degradation. Thus, a careful trade-off analysis is necessary, weighing the gain due to transmission time reduction versus the loss due to computational overhead and data quality degradation. This chapter examines the objectives of the distributed visualization environment, and develops an objective function for optimization.

The following symbol convention is adopted for this discussion.

- symbol names
  - C : compression rate
  - D : data
  - O : operation

  - S : speed
  - T : time
  - V : data volume

- subscripts
  - N : Network
  - H : Host (Visualization Server) system
  - U : User (Client) system
  - c : compression, compressed
  - cv : compressed data visualization
  - ct : compressed data transmission
  - d : desired
  - dc : decompression, decompressed
  - r : raw
  - rv : raw data visualization

- indices
  - l : decomposition level
  - b : bit plane
  - p : processor

2.1 Objectives

In order to satisfy the requirements of a distributed visualization environment, one's data compression approach must achieve the following two objectives.

Objective 1: The time involved in compressed data visualization must be significantly less than that of raw data visualization.

$$T_{cv} \ll T_{rv} \tag{1}$$

Objective 2: The loss introduced by data compression (the difference between the original data and the restored data) must be acceptable.

$$D_r - O_{dc}(D_c(C)) < \epsilon \tag{2}$$

2.2 Implementation Criteria

In order to optimize the performance of a data compression, the following three implementation criteria may be observed.

Criterion 1: The time for compression, transmission, and decompression may be balanced so that the execution times can be pipelined optimally. In a pipelined implementation, the time involved in compressed data visualization is the maximum of the three time values.

$$T_{cv} = MAX(T_c, T_{ct}, T_{dc}) \tag{3}$$

The time for compression and decompression is a function of the compression speed of the respective computational platform and the operational complexity involved.

$$T_c = O_c/S_H \tag{4}$$
$$T_{ct} = V_c/S_N \tag{5}$$
$$T_{dc} = O_{dc}/S_U \tag{6}$$

Criterion 2: The time for compression or decompression may not be longer than the time for transmission; so the time involved in compressed data visualization can be represented as the time for transmission only.

$$T_{ct} \geq MAX(T_c, T_{dc}) \tag{7}$$

$$T_{cv} = T_{ct} \tag{8}$$

Criterion 3: The second criterion implies that there is no benefit in reducing the data volume beyond that which can be transferred within the time required to either compress or decompress the data, whichever is higher.

$$V_c \geq S_N MAX(T_c, T_{dc}) \tag{9}$$

2.3 Objective Function

An optimal application of the compression technique in a distributed visualization environment can be formulated as an objective function where the optimal solution is to maximize the objective function. An objective function is suggested as a gain function expressed below, where the gain is computed as a weighted sum of the time savings in data transfer plus the amount of data degradation.

$$Gain = w_1(T_{rv} - T_{cv}) + w_2(-|D_r - O_{dc}(D_c(C))|) \tag{10}$$

The weighting factors may be dependent on applications and operational situations. For example, when a user is interested in quick browsing, the data quality degradation may not matter as much as the transmission speed. Also, due to the data dependency of the relationship between the compression rate and quality degradation, it is difficult to provide fixed values for the optimal trade-off between the compression rate and the data loss.

3 Prototype System

A prototype system was developed employing a client/server model where compression is performed on a visualization server and decompression and display functions are performed on a client. The prototype system employs a Cray T3D as a server system and a UNIX workstation as a client. The compression function was implemented to support two application modes: quick browse and progressive transmission.

The quick browse mode compresses data for a desired compression rate while the progressive transmission mode compresses data losslessly and allows for progressive retrieval of the compressed data for any arbitrary compression rate. An additional feature of the progressive transmission mode is in compression rate controllability. This section discusses the two operation modes for their application areas, examines the steps involved in parallel compression algorithm implementation, and describes the interface mechanism for integrating the compression with visualization application programs.

3.1 Operation Modes

Both quick browse mode and progressive mode utilize Fast Wavelet Transform (FWT)-based subband coding compression algorithm. The basic steps involved in the compression algorithm are FWT, quantization, run-length encoding, and Huffman encoding. In the quick browse mode, the quantization is applied once with a step size appropriate for the desired compression rate. The one-step quantization can be compared to discarding the bits below the quantization step size. The discarded bits represent the information loss (which is referred to as data quality degradation). Due to the coarse quantization step size, the compression rate cannot be controlled precisely.

In the progressive transmission mode, the transformed coefficients are decomposed into bit planes and each bit plane is encoded independently. The bit plane decomposition is equivalent to applying the quantization step iteratively to the transformed coefficients where the quantization step size is reduced by a factor of two per iteration. When the compression rate is specified, the encoded bit planes are counted from the highest bit plane until the count reaches the specified compressed data size. The bit plane ordering allows precise control of the compression rate with a minimum of information loss. A detailed description is available in the Bit-Plane Encoding section of the ERIC algorithm report (in Part II).

The advantage of the quick browse mode is in the computational simplicity (one-step quantization), while the advantage of the progressive transmission mode is in the flexible retrieval of the compressed data. An optimal usage of the two modes in an interactive visualization environment may be coupling them to the viewer's motion speed where during a fast fly-over period, the quick browse mode is employed and as the viewer slows down the mode is switched to the progressive transmission mode. As a viewer stops at a location, the location data may be transmitted in full resolution progressively.

3.2 Parallel Implementation

The four basic steps in the compression algorithm—FWT, quantization, run-length encoding, and Huffman encoding—were examined for parallelism by analyzing operational locality, homogeneity, and load balancing.

Initially, a complete domain decomposition approach was employed, where a dataset is equally divided among the processors and each subarea is independently compressed. This approach was found to be sub-optimal with respect to compression rate and data quality. The overhead of each processor having its own Huffman table became very large as the number of processors increased, reducing the maximum achievable compression rate. The data quality was degraded further due to the inter-processor data quality variation. The variation was introduced by applying equal compression rates among processors; it also stemmed from the discontinuity at segment boundaries.

To correct the Huffman table overhead problem, a single-Huffman-table approach was adopted. In order to use one Huffman table among the processors, the histograms of the run-length code from each processor need to be merged to generate a global histogram. From the global histogram, each processor builds a Huffman table and generates Huffman codes. Since the Huffman table is the same for all nodes, only one Huffman table needs to be transmitted to the decompressor. The time introduced by the histogram merging step was found to be insignificant.

To correct the inter-processor data quality variation in the fixed compression rate, the compressed data size information per bit plane is shared among the processors so that the compression rate is applied globally. The total compressed data size was compared to the total compressed bit plane size to compute the number of bit planes that can be transmitted whole; for the bit plane that needs to be transmitted partially, the partial amount per processor was computed based on equal rationing. The formula can be expressed as below:

$$V_d = V_r / C_d \tag{11}$$

$$V_x = \sum_{p=1}^{p=P} \sum_{b=B}^{b=x} D_c(b,p) \;<\; V_d \tag{12}$$

$$V_y = V_d - V_x \tag{13}$$

$$V_z = \sum_{p=1}^{p=P} D_c(x-1,p) \tag{14}$$

$$V(q) = (V_z/V_y) D_c(x-1,q) \tag{15}$$

where V_d is the desired data volume, V_x is the data volume of the bit planes that can be transmitted as a whole, V_y is the data volume available for the $(x-1)$-th bit plane, V_z is the data volume of the $(x-1)$-th bit plane, $V(q)$ is the data volume allowed to be transmitted per processor q for the $(x-1)$-th bit plane, P is the number of processors employed for compression, and B is the highest bit plane number.

The boundary discontinuity is caused by the artificial data assumed by each processor during the wavelet transform process at the boundary. The artificial data assumed at the boundary yields a different level of loss at each boundary between adjacent processors when the transformed coefficients are quantized. The different levels of loss imply different levels of restorability, which contributes to the quilted appearance when the image is decompressed.

To correct the boundary discontinuity problem entirely, the real data must be used during the wavelet transformation for the entire subarea so that the transformed coefficients computed by the multiple processors are the same as those computed by a single processor. However, such provision requires a very large amount of overlap between processors. For the prototype, the overlap size was empirically determined by observing the detectability of the distortion as well as RMS error. The overlap was not necessary for a compression rate less than 16; and one or two lines were found to be sufficient for a compression rate less than 100.

3.3 Application Program Interface

When the compression and decompression functions are integrated in an application program, the application program must establish the network connection between the server and the client, distribute the data among the processors, initialize the compression function, and apply compression per frame in a loop. The number of processors employed for compression may be a subset of the processors involved in the visualization application, in order to meet the minimum subblock size constraint.

- Network Connection : The compression function can return the compressed data from individual nodes as well as from node 0 after merging the results from all nodes. The individual transmission may provide an additional speed up when the compression time varies significantly among the nodes. For each node that transfers the compressed data to the host system, a socket connection must be established.
- Initialization : During initialization, the following information on the data must be provided in order to allocate necessary memory, reformat the data, set up appropriate compression parameters, and initialize the decompressor.
  - socket ID
  - global data size
  - subarea size
  - number of processors employed for compression per spectral channel
  - number of spectral channels of data (1 for B/W, 3 for color, 4 for color + overlay)
  - desired compression option (0 for no compression, 1 for browse(default), 2 for progressive)
  - desired compression rate (integer greater than 1)
  - result merging option (0 for no merging (default), 1 for merging: when merging option is not set, the socket IDs must be provided for all nodes)

- Compression : Per frame, the starting offset and memory location of the subarea are passed to the compressSend() function. The compressSend function performs data preparation, compression, and transmission of the compressed data to the decompressor through the socket ID.

- Reinitialization : Only the compression rate may be reinitialized between compressions.

4 Performance Analysis

The performance of the parallel distributed visualization system employing data compression was investigated with respect to execution time, data degradation, and data volume reduction. The investigation was performed by benchmarking the above information for various compression option settings including number of processors, decomposition level, compression rate, etc. The benchmark timing was obtained on the Cray T3D for compression and SGI Indigo 2 Extreme (IRIX 5.3) for decompression and display, using a 512×512 gray-scale image of Lena.

4.1 Compression Parameters

There are four parameters that affect compression results in terms of execution time, achieved compression rate, restored data quality, and required boundary area overlap. They are

- nl_{lh} : number of decomposition levels for which both low-pass and high-pass coefficients are computed

- nl_{low} : number of decomposition levels for which only low-pass coefficients are computed

- Q : quantization factor which is applied to quantize the transformed coefficients

- $overlap$: size of overlapped area at segment boundary

The minimum subarea size per processor (n_{min}), which is a lower bound in each dimension, is a function of the decomposition levels and the wavelet filter size as defined below. (In the prototype, a 5×3 wavelet filter was employed.)

$$n_{min} \geq 2^l + 2 \tag{16}$$

$$l = nl_{low} + nl_{lh} \tag{17}$$

The quantization factor plays the major role in determining the resulting compression rate and noise. The other parameters provide minor improvements in some special cases. For example, the reason for allowing a decomposition level for which only low-pass coefficients are computed is to reduce execution time when high compression rate is specified assuming that the high-pass coefficients can be ignored. Thus, the parameter is irrelevant in the cases with low compression rate.

The achievable compression rate was analyzed for four quantization factors (32, 64, 128, 256) with no low-pass-only level ($nl_{low} = 0$) and with one low-pass-only level ($nl_{low} = 1$).

These eight cases provided a guideline to an appropriate mapping between compression rate and the parameter setting. The overlap amount was adjusted to achieve compatible visual quality at boundaries between the runs employing different number of processors.

Part III.A shows the plots of compression/decompression time as a function of the number of processors employed for computation. Times with and without the use of overlap are plotted. These figures also show plots of the compression rate obtained with and without the use of overlap. A way to avoid using too large an amount of overlap is to not discard the overlapping area, but instead perform an averaging operation in the area of overlap. This method attempts to smooth the boundary transition, thereby reducing the amount of overlap used.

4.2 Parallel Versus Sequential

The parallel version introduces three types of computational overheads compared to the sequential version: data distribution, global histogram generation, and data merging. It also introduces two types of performance degradation: compression rate reduction, and larger noise due to the shortened run length and increased number of boundaries as the data is segmented to many subareas. In order to understand the computational overhead and performance degradation, a detailed benchmark was performed.

Details of the benchmark results are presented in Part III.B. For each benchmark run, the compression and decompression functions were executed 10 times in a loop. On the compression side, each timing result represents an average value (in seconds) whereas on the decompression side, each timing result represents the minimum value.

For the computational overhead analysis, the transform, quantization, and encoding steps were benchmarked for the five cases of different number of processors. The Entropy Encode step was examined for run-length coding and Huffman coding. And the Huffman coding step was further investigated for histogram generation, global histogram generation, Huffman table generation, and Huffman encoding. The timing distribution demonstrates that the computation time reduces linearly as the number of processors increases for the transform, quantization, and run-length encoding, indicating no or negligible overhead. The detailed timing analysis of the Huffman coding step indicates a constant overhead of 0.05 seconds for global histogram generation and Huffman table generation.

For the performance degradation analysis, the run-length code size (RLE datasize), Huffman table size (HTable size), compressed code size (HE datasize), and noise level (RMS error) were benchmarked for the five cases of different numbers of processors. From the list, it can be noted that the compression rate reduction of approximately 0.25 per additional processor is necessary to prevent an increase in the level of noise.